MASTER'S THESIS 2019:NN

Self-stabilizing Media Access in Sensor Networks

ZAHID IQBAL

Department of Computer Science and Engineering
Division of Computing Science
Distributed Computing and Systems Research Group
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2019

Self-stabilizing Media Access in Sensor Networks

Master's Thesis 2019:NN

Department of Computer Science and Engineering
Division of Computing Science
Distributed Computing and Systems Research Group
Chalmers University of Technology
SE-41296 Göteborg
Sweden

Tel. +46-(0)31 772 1000

Department of Computer Science and Engineering
Göteborg, Sweden 2019

Abstract

We consider the problem of accessing a single shared media in wireless sensor networks. We assume a fixed communication graph and a globally synchronized common pulse but no clock synchronization. We implement a silent self-stabilizing algorithm that allocates time slots in which the nodes broadcast. We empirically demonstrate that within a constant number of communication rounds, a large number of nodes communicate successfully, without collisions. Moreover, the ratio between successful and unsuccessful broadcasts rapidly approaches to 1. The experiments were carried out using the TOSSIM simulator.

Key words: self-stabilization, random geometric graphs, extended degree, communication graph, interference graph

Acknowledgements

This work would not have been completed without help and support of many individuals. I would like to thank everyone who has helped me along the way. Particularly: Dr. Elad Michael Schiller for providing me an opportunity to conduct my master's research with him and for his guidance and support over the course of my thesis, Andreas Larsson for helping me with various technical details in the initial phases of the project, Dr. Philippas Tsigas for serving on my thesis committee, my present and past roommates for all the memorable times and lastly, my family without whose support none of this would have been possible.

Contents

1 Introduction

The task in this graduate work has been to develop a media access protocol for wireless sensor networks. In wireless sensor networks, there is a single shared medium. The goal is to have sensor nodes broadcast on such medium without collisions in a fixed number of communication rounds. This thesis is motivated by the fact that energy is the scarcest source in wireless sensor nodes, and best way to efficiently utilize this energy is to have efficient media access protocols that are self-stabilizing and guarantee collision free broadcasts. The implementation of the protocol is carried out in nesC which is a language intended for embedded systems. The implementation is tested using TOSSIM (TinyOS Simulator).

The following description in this section would provide an overview of the wireless sensor networks, their importance, key components making up a wireless sensor device, and design and implementation challenges while developing a wireless sensor network application.

A wireless sensor network is an infrastructure comprised of large numbers of spatially distributed, low-power, inexpensive and autonomous sensor devices operating together in a wireless network. The envisioned applications of wireless sensor networks range widely: ecological habitat monitoring, structure health monitoring, environmental contaminant detection, industrial process control, and military target tracking among others.

Wireless sensor networks provide bridges between the virtual world of information technology and the real physical world. They represent a fundamental paradigm shift from traditional inter-human personal communications to autonomous inter-device communications. They promise unprecedented new abilities to observe and understand large-scale, real-world phenomena at a fine resolution. As a result, wireless sensor networks also have the potential to engender new scientific advances. Depending on the application, WSN devices can be networked together in a number of ways. In basic data-gathering applications, for instance, there is a node referred to as the sink to which all data from source sensor nodes are directed. The simplest logical topology for communication of gathered data is a single-hop star topology, where all nodes send their data directly to the sink. In networks with lower transmit power settings or where nodes are deployed over a large area, a multi-hop tree structure may be used for data-gathering. In this case, some nodes may act both as sources themselves, as well as routers for other sources.

Embedded wireless sensor networking devices called motes were developed by researchers at Berkeley. These were made publicly available commercially along with TinyOS, an associated embedded operating system that facilitates the use of these

devices. The ongoing wireless networks revolution can be attributed to the availability of these devices as an easily programmable, fully functional, relatively inexpensive platform for experimentation.

1.1 Networked Wireless Sensor Devices

As shown in Figure 1.1, there are several key components that make up a typical wireless sensor network device; a *low-power embedded processor* to carry out computational tasks on a WSN device that includes processing of both locally sensed information as well as information communicated by other sensors. At present, primarily due to economic constraints, processors are significantly constrained in terms of computational power i.e. 8 bit, 16 MHz processor. It is due to these constraints that devices typically run a specialized component based embedded operating system, such as TinyOS. *Memory* includes the program memory and data storage which are constrained due to economic considerations and are likely to improve over time. *Radio transceiver* in a WSN device comprises a low-rate, short-range wireless radio (10–100 kbps, <100m). Radio communication is often the most power-intensive operation in a WSN device, and hence the radio must incorporate energy-efficient sleep and wake-up modes. Due to bandwidth and power constraints, WSN devices primarily support only low-data-rate sensing. Each device may have several *sensors* on board depending on the application; for example, temperature sensors, light sensors, humidity sensors, pressure sensors, chemical sensors, acoustic sensors, or even low-resolution imagers. For flexible deployment WSN device is likely to be battery powered [1]. The finite battery energy happens to be the most critical resource bottleneck in most WSN applications.

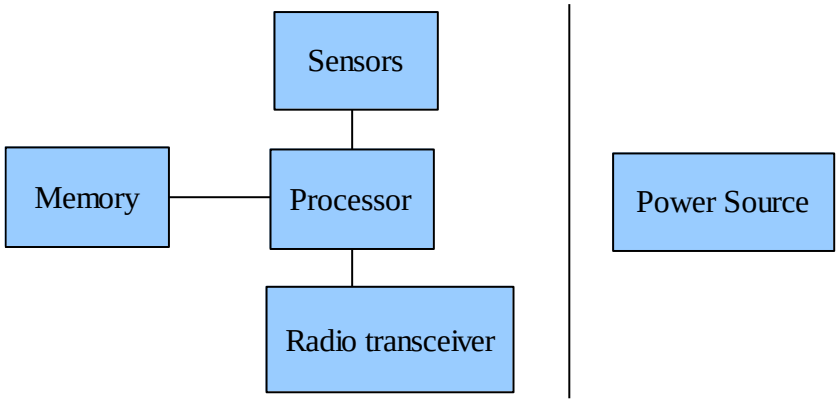

Figure 1.1: *Schematic of a basic wireless sensor network device*

1.2 Key Design Challenges

Wireless sensor networks are interesting from an engineering perspective, because they present a number of serious challenges that cannot be adequately addressed by existing technologies. Below, we discuss some of the design challenges that are addressed by this thesis work.

Extended lifetime: WSN nodes will generally be severely energy constrained due to the limitations of batteries. A typical alkaline battery, for example, provides about 50 watt-hours of energy; this may translate to less than a month of continuous operation for each node in full active mode. Due to potential infeasibility of monitoring and replacing batteries for a large network, much longer lifetimes are desired. In practice, it will be necessary to provide guarantees that a network of unattended wireless sensors can remain operational without any replacements for several years. Hardware improvements in battery design and energy harvesting techniques will offer only partial solutions. This is the reason that most protocol designs in wireless sensor networks are designed explicitly with energy efficiency as the primary goal. Since with self-stabilizing media accesses, there are successful broadcast and collisions avoidance thus leading to energy efficiency in saturated situations.

Responsiveness: The network lifetime can be extended by having the nodes operate in a duty-cycled manner with periodic switching between sleep and wake-up modes. While synchronization of such sleep schedules is challenging in itself, a larger concern is that arbitrarily long sleep periods can reduce the responsiveness and effectiveness of the sensors. In applications where it is critical that certain events in the environment be detected and reported rapidly, the latency induced by sleep schedules must be kept within strict bounds, even in the presence of network congestion.

Robustness: The use of large numbers of inexpensive devices in wireless sensor networks is motivated by the vision to provide large-scale, yet fine-grained coverage.

However, inexpensive devices can often be unreliable and prone to failures. Also, in harsh or hostile environments, the rates of device failure will be high. Protocol designs must therefore have built-in mechanisms to provide robustness. Further, it is often desirable that the performance of the system degrade as gracefully as possible with respect to component failures. Under current settings, the developed system assumes globally synchronized pulses and stabilizes. Future work would consider to extend the algorithm for most realistic settings of fluctuating clock skews but still offering stabilization and hence robustness of the wireless sensor network.

Scalability: Wireless sensor networks have the potential to be extremely large scale (tens of thousands, perhaps even millions of nodes in the long term). Protocols will have to be inherently distributed, involving localized communication, and sensor networks must utilize hierarchical architectures in order to provide such scalability. However, visions of large numbers of nodes will remain unrealized in practice until some fundamental problems, such as failure handling and in-situ reprogramming, are addressed even in small settings involving tens to hundreds of nodes. Under the current limitation of simulation, we have managed to test the algorithm with varying sizes of network and the performance of the algorithm remains consistent.

Self-configuration: Because of their scale and the nature of their applications, wireless sensor networks are inherently unattended distributed systems. Autonomous operation of the network is therefore a key design challenge. From the very start, nodes in a wireless sensor network have to be able to configure their own network topology; localize, synchronize, and calibrate themselves; coordinate inter-node communication and determine other important operating parameters.

Self-optimization and adaptation: Traditionally, most engineering systems are optimized a priori to operate efficiently in the face of expected or well-modelled operating conditions. In wireless sensor networks, there may often be significant uncertainty about operating conditions prior to deployment. Under such conditions, it is important that there be in-built mechanisms to autonomously learn from sensor and network measurements collected over time and to use this learning to continually improve performance. Also, besides being uncertain a priori, the environment in which the sensor network operates can change drastically over time. WSN protocols should also be able to adapt to such environmental dynamics in an on line manner. With some ideal assumptions, the protocol self-stabilizes when started in arbitrary configurations. We aim to extend the same algorithm for self-stabilization under more realistic scenarios.

Synchronization: also becomes a problem within sensor networks since the requirement for low cost devices often necessitates the use of lower precision hardware. The developed protocol assumes globally synchronized common pulse for all the nodes. Therefore, future work entails providing self-stabilization using realistic clock values.

1.3 Report Organization

This report will start by introducing the main issues and problems regarding wireless communication in sensor networks in chapter 2, followed by a discussion of existing medium access protocols for sensor networks in chapter 3. In Chapter 4, we discuss the related work in the area of self-stabilizing media access in sensor networks and compare it with our developed approach. Chapter 5 gives the thesis statement, and provides the pseudo code of the developed algorithm. The implementation issues and improvements in the algorithm have been presented in the form of a revised algorithm in the chapter 6. The following chapter details the tools used for the project and the modifications we had to incorporate to the existing tools. Chapter 8 presents the experiment settings, and details the results of preliminary numerical evaluation. The report ends in chapter 9 with a discussion regarding the obtained results, conclusions made, and future work suggested.

2 The Problems

Wireless communication is the key to the flexibility of sensor networks and their low cost deployment. However, wireless link conditions vary due to multi-path propagation effects. This offers a significant challenge to the performance of protocols designed for sensor networks. In the Ideal Model, two nodes have a perfect link (100 % packet reception rate), if they are within communication range R of each other, and a non-existent link (0% packet reception rate) if they are outside the reception range. This model, while simple to implement and reason about, has little basis in reality [3].

Empirical observations reveal that the packet reception contour formed by receptions at different locations from the same transmitter is not isotropic. Link quality distributions are highly dependent on environmental and individual hardware differences. Indoor office environments, for instance, show worse link quality distributions than clutter-free outdoor settings. There are three distinct regions of link quality; a nearby connected region where packet reception rates are consistently high, beyond this lays a transitional region where reception rates are highly variant, and beyond that there is a disconnected region where transmissions from this node cannot be heard. In the transitional region, there can be links that have excellent quality, although the node pairs are relatively far apart, and conversely there can be weak links that have poor quality, despite the relative proximity of the node pairs [1]. Hardware variations cause node pairs to have different SNR curves, but for any given pair the curve is precise [3]. Experimental studies indicate strongly that the specific hardware combination of sender and interferer change the measured SINR threshold (signal to interference plus noise ratio). Most research regarding network interference assumes one of two interference models: protocol model or the physical model. In protocol model which is implemented by many simulators, concurrent transmissions from any sender within a given range (referred to as the interference range) of receiver will cause a collision that results in the loss of packet from the corresponding sender. The physical model considers "the capture effect", also called co-channel interference tolerance, which is the ability of certain radios to correctly receive a strong signal from one transmitter despite significant interference from other transmitters. With capture effect, simultaneous successful receptions are possible so long as SINR is sufficiently high at each receiver [5].

Wireless sensor networks often have the hidden node problem and the exposed node problems. The hidden node problem is illustrated in Figure 2.1 (a); here, node A is transmitting to node B. Node C, which is out of the radio range of A, will sense the channel to be idle and start packet transmission to node B too.

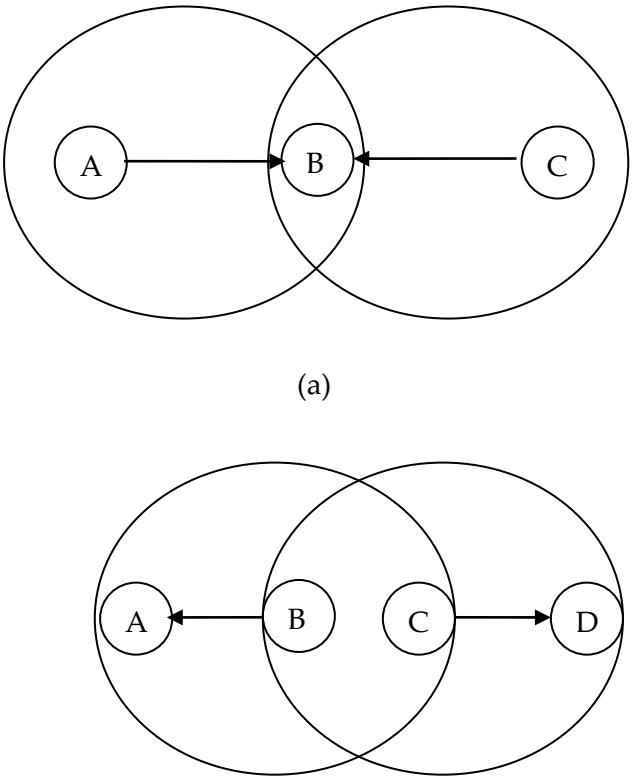

Figure 2.1: *Problems in wireless environment: (a) hidden node (b) exposed node*

The exposed node problem is illustrated in Figure 2.1 (b). Here, while node *B* is transmitting to node *A*, node *C* has a packet intended for node *D*. Because node *C* is in range of *B*, it senses the channel to be busy and is not able to send. However, in theory, because *D* is outside of the range of *B*, and *A* is outside of the range of *C*, these two transmissions would not collide with each other. The deferred transmission by *C* causes bandwidth wastage. These problems are duals of each other in a sense: in the hidden node problem packets collide because sending nodes do not know of another ongoing transmission, whereas in the exposed node problem there is a wasted opportunity to send a packet because of misleading knowledge of a non-interfering transmission.

In the presence of aforementioned problems, it is not straightforward to avoid collisions in wireless sensor networks.

3 Background

Limited energy, computational, and communication resources complicate the protocol design within sensor networks. Constraints on sensor node cost further restrict which technologies sensor network may utilize. The goal of long term, independent operation of large scale sensor networks under such restrictions is yet to be achieved. Medium access protocols provide the greatest influence over communication mechanisms and hence the utilization of the transceiver, the largest energy consumer in most sensor nodes [4].

3.1 Existing Solutions

An essential characteristic of wireless communication is that it provides an inherently shared medium. All medium-access control (MAC) protocols for wireless networks manage the usage of the radio interface to ensure efficient utilization of the shared bandwidth. MAC protocols designed for wireless sensor networks have an additional goal of managing radio activity to conserve energy. Thus, while traditional MAC protocols must balance throughput, delay, and fairness concerns, WSN MAC protocols place an emphasis on energy efficiency as well.
We have to solve multiple access issues. There are two different types of protocols to resolve such issues: random access protocols and channelization protocols. In random access or contention methods, no station is superior to another station and none is assigned the control over another. No station permits, or does not permit, another station to send. At each instance, a station that has data to send uses a procedure defined by the protocol to make a decision on whether or not to send. Channelization is a multiple access method in which the available bandwidth of a link is shared in time, frequency or through code among different stations.
We present a discussion of medium access control concepts in relation to sensor networks and examine previous wireless medium access control protocols to illustrate how they do not match the requirements and characteristics of sensor networks.

3.1.1 CSMA and CSMA/CA

Carrier Sense Multiple Access (CSMA) is the simplest form of medium access. It has two variations: non-persistent CSMA, and p-persistent CSMA. In non-persistent CSMA, a station that wishes to transmit senses the channel to see if it is idle. If it finds the channel busy, it would perform a random back off by waiting before

attempting to transmit again. When it finds the channel idle, it would transmit immediately. In p-persistent CSMA, the station would continue to sense the busy channel instead of delaying and checking again later. When the channel is available, the station transmits with probability p and defers the transmission with probability 1-p. But constant channel sensing prevents sensor nodes to use CSMA without modifications because the transceiver consumes energy too quickly. Also, when two or more stations choose transmission times that are close, collisions are bound to happen.

CSMA with collision avoidance (CSMA/CA) is an extended version of CSMA. Wireless network attempt to avoid collisions instead of detecting them. One reason is that data corruption due to collision occurs at the receiver and the sender can not know of a failed transmission. Another reason is that in wireless LANs it is not possible to listen while sending; therefore collision detection is not possible. Adding a full duplex transceiver or a second half duplex transceiver would increase the monetary and energy costs, and complicate the device design.

CSMA/CA uses control messages to reserve the link. The sender who wishes to transmit first performs basic CSMA to find an appropriate transmission time and then transmits an RTS (request to send) control message to the intended receiver. The receiver can reply with CTS (clear to send) message. The reception of a CTS indicates that the receiver is able to receive the RTS, so the packet (the channel is clear in its area). At the same time, every node in the range of the receiver hears the CTS (even if it doesn't hear the RTS), so understands that a transmission is going on. The nodes hearing the CTS are the nodes that could potentially create collisions in the receiver. All nodes avoid accessing the channel after hearing the CTS even if their carrier sense indicates that the medium is free. RTS/CTS have another advantage: it lowers the overhead of a collision on the medium (collisions are much shorter in time). If two nodes attempt to transmit in the same slot of the contention window, their RTS collide and they don't receive any CTS, so they loose only a RTS, whereas in the normal scenario they would have lost a whole packet.

The RTS/CTS handshaking adds a significant overhead for sensor networks where data messages have sizes comparable to control messages.

3.1.2 CSMA/CD

CSMA with collision detection (CSMA/CD) is another extended version of CSMA. Collision detection is used to improve CSMA performance. The sender first performs the basic CSMA to find an appropriate transmission time, and then starts transmitting the frame. A transmitting station that detects another signal while transmitting a frame, stops transmitting that frame, transmits a jam signal, and then waits for a random time interval (known as "back off delay") before trying to send that frame again. Back off delay is determined using truncated binary exponential

back off algorithm. CSMA/CD is used on wired network like Ethernet because on a wire, the transceiver has the ability to listen while transmitting and so to detect collisions (with a wire all transmissions have approximately the same strength). But, even if a radio node could listen on the channel while transmitting, the strength of its own transmissions would mask all other signals on the air [2]. So, the protocol can't directly detect collisions like with Ethernet.

3.1.3 Receiver-side Collision Detection

Receiver-side collision detection is a better suited technique for wireless sensor networks with the message collisions being detected at the receiving node. Some works have made use of receiver side collision detection techniques. [7] Employs this idea to propose a reliable single hop broadcast solution for sensor networks whereas [8] uses this technique to offer a fault-tolerant solution to the consensus problems in the presence of crash-prone sensor nodes. The collision detector in [7] and [8] works as follows: For every round r of each execution, if a node p does not receive some messages that were broadcast in r, then p detects a collision in r.

3.1.4 TDMA

Time division multiple access (TDMA) is a channel access method for shared medium networks. A specific node, the base station, has the responsibility to coordinate the nodes of the network. The time on the channel is divided into time slots, which are generally of fixed size. Each node of the network is allocated a certain number of slots where it can transmit. Slots are usually organized in a frame, which is repeated on a regular basis. TDMA protocols provide collision free medium access. However such a system requires efficient time synchronization for the entire network. Changes in the network topology require a modification in the schedule or slot allocation. Finally, static allocation of slots can leave many slots unused reducing the throughput of the network. In TDMA if a user does not have any data to send, no other user can use their slot. Hence, the full capacity of the Broadcast network or channel is not fully being exploited.

4 Related Work

We discuss the related work that has been done in the area of self-stabilizing media access in sensor networks.

Herman and Tixueill [6] present a self-stabilizing algorithm for distributed slot-allocation in sensor networks. Their algorithm avoids collision as in TDMA but offers additional features of stabilization in the event of transient faults and topology changes. This TDMA slot assignment is the vertex colouring problem where we ensure that no pair of vertices at distances two or less has the same colour [6]. All nodes share clocks that are synchronized to a common global time. The communication is half duplex; a node cannot transmit one message and receive another message concurrently. Therefore collisions are possible. CSMA/CA is used to avoid collisions: if node p has some message ready to transmit, but is receiving some signal, then p does not begin transmission until it detects the absence of signal.

Each node is assigned a colour and the colours are used as a schedule for TDMA slot assignment. Before the colour assignments, radio time is partitioned into two parts: one part is for TDMA scheduling of application messages known as the TDMA part, and other part is for the messages of the algorithm that assigns colours to nodes known as the overhead part. Figure 4.1 shows such arrangement. CSMA is used to manage collisions in overhead part. TDMA slots do not use random delay and nodes use relevant slots without collisions.

Simple distance two colouring algorithms may use a large number of colours that is wastefully large. One approach to use a reasonable number of colours is to choose a set of leader nodes who dictate the assignment of colours to nodes. This colouring is minimal. Leaders are chosen through a maximal independent set. An independent set is a set of vertices in a graph no two of which are adjacent. A maximal independent set is a set such that adding any other vertex to the set forces it to contain an edge; in other words, a maximal independent set is the largest independent set of a graph.

We consider a saturated situation where every node is allocated a single time slot and data grams are always available. Since, CSMA/CA is used in the overhead section, which leaves the possibility of longer stabilization time and unfair bandwidth allocation. We don't use the overhead section, the overhead of our developed approach is lower after stabilization and there remains a fair bandwidth allocation to all the nodes.

The work of Demibras and Balachandran [7] addresses the hidden node problem and proposes single hop reliable broadcast solution for sensor networks. They use receiver side collision detection [8] and assume globally synchronized rounds where

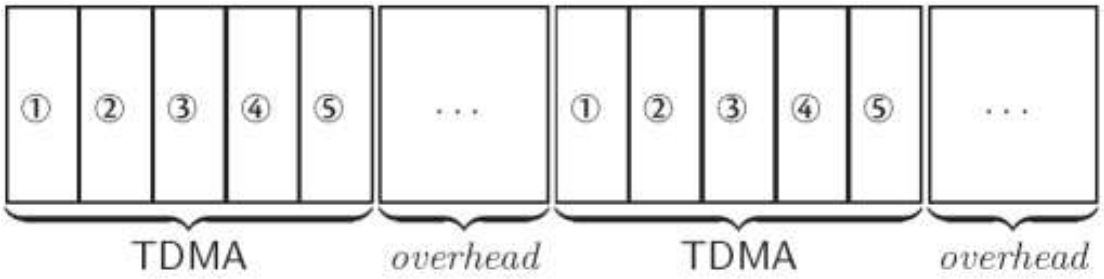

Figure 4.1: *Scheme of radio time partition in Herman and Tixueill design*

each round has three phases: RTS (request to send), NCTS (not clear to send) and Data phases. A node j wishing to transmit data does so by sending a request in RTS phase. Neighbouring nodes respond to an invalid RTS or collided RTS's by transmitting a not-clear-to-send message (NCTS) during the NCTS phase. The neighbours are able to sense collisions at the RTS phase using receiver side collision detection techniques. The node j backs off from transmitting the data for this round if it either receives a NCTS message or detects a collision in the NCTS phase. This scheme avoids potential collision of data during the Data phase, and ensures reliable delivery of the payload within single-hop distance of the transmitting node. Figure 4.2 shows the arrangement.

They contend that at any time only one node in single hop neighbourhood is in the Data phase. Hence, there is no possibility of a node receiving concurrent transmissions from two nodes which are hidden from each other. We consider a saturated situation where data grams are always available. In such a situation, there will be significant collisions in RTS phase. This would let the transmitters back off and thereby lead to delays. It is not clear, if the back off time is random since a random back off time leaves a possibility of longer stabilization time. Moreover, we see that saturated situations would incur large overheads even after the network has been stabilized and every time a significant delay would be involved before stabilization. Again, our protocol performs better under such scenario; stabilization is quick and there are only low overheads after stabilization.

Figure 4.2: *Synchronized rounds in Demibras and Balachandran design*

5 Thesis

We develop a distributed media access algorithm and we empirically demonstrate that within a constant number of communication rounds, a large number of nodes communicate successfully, i.e. without collisions. Moreover, the ratio between successful and unsuccessful broadcasts rapidly approaches to 1. We employ the technique of receiver-side collision detection. In receiver side collision detection, the receiving node reports the collision to all the neighbouring nodes. We consider the pros and cons of such an algorithm:

- *Wait at least 1 round before detection is delivered*

  The node chooses a new slot when there is collision but it skips one round. Thus it has to wait for the duration of a round before broadcasting in the chosen slot.

- *The algorithm can emulate pair wise independent choices*

  Each node chooses a time slot uniformly at random independently of the neighbouring nodes. While there remains the probability of two or more nodes selecting the same time slot to broadcast and thus causing collisions. But eventually, all nodes broadcast successfully since after the collision, the nodes make a new random choice for the broadcast time slot.

- *There is no need for clock synchronization*

  Our algorithm assumes that the clocks of all the nodes are synchronized to a common global pulse.

- *This presents a more realistic CSMA/CD with current HW*

  The CSMA/CD algorithm is not feasible with current HW of sensor nodes as a node cannot listen to the medium while it is transmitting. Since the collision detection in our algorithm is carried on the receiver side, our simulation presents a more realistic solution for the sensor devices.

- *We have lower overheads after stabilization*

  There are little or no overheads after stabilization due to self-stabilizing nature of the algorithm.

- *The algorithm presents a more decentralized and robust solution*

There is no central node to assign bandwidth. Each node makes its own bandwidth allocation based on the slot information of neighbouring nodes in the communication graph. If a transient fault causes two or more nodes to send messages concurrently over the shared media, collisions would occur. In such an event, the algorithm ensures that the system rapidly returns to a stabilized state where no collisions occur.

Tiny artifacts use wireless communications over a shared media, e.g., radio broadcasting, in which messages collide when sent concurrently. We assume the existence of an ideal environment that has access to synchronized clocks and perfect collision detectors. We assume that the system is synchronous and that all processors share a common source of global pulses. We call the period between two pulses as time slot, and assume that any message sent during a particular time slot is received at most once within that time slot, but not in any other time slot. We assume that when two neighbouring processors broadcast concurrently, their messages collide. If two processors don't share an immediate neighbour then they can broadcast concurrently. Immediate neighbours of a particular processor are at distance one from that processor.

Before presenting the algorithm, it may be helpful to consider the relationship between the slot assignment to nodes and the standard problem of graph colouring. Algorithmic research relates the problem of slot assignment to minimal graph colouring where the colouring constraint states that no two nodes within distance two have the same colour. This constraint comes from the well known hidden terminal problem in sensor networks. This leaves us to choose D (upper bound on the round size) to be not the size of communication graph but a number represented by maximum of node plus immediate neighbours plus neighbours of immediate neighbours taken uniquely referred to as the second or extended degree in our discussions.

We look into what algorithms exist for distance-two unique time slot allocation.

5.1 Deterministic Sequential Algorithm

The deterministic sequential algorithm can be represented in the following way:

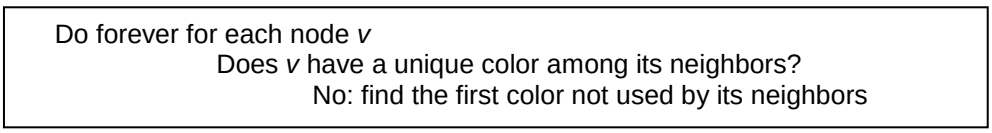

Figure 5.1: *Deterministic Sequential Algorithm*

Here, we assume that the colours are ordered in sequence and a particular vertex v gets the colour not already assigned to its neighbours. As sensor nodes are deployed in large numbers, it may not be possible to collect the entire topology at a single node and apply the algorithm. The sequential algorithms are not fast enough in the distributed world.

5.2 Randomized Sequential Algorithm

The randomized sequential algorithm can be represented in the following way:

```
Do forever for each node v
        Does v have a unique color among its neighbors?
                No: select a color uniformly at random from an entire
                set of colors that are not used by neighbors
```

Figure 5.2: *Randomized Sequential Algorithm*

These algorithms employ a degree of randomness in their logic. These algorithms can quickly assign unique colours with high probability. This algorithm performs better than deterministic sequential algorithm but is not good in deployments of very large number of sensor nodes due to its sequential nature.

5.3 Our Algorithm: Randomized Distributed Algorithm

The randomized distributed algorithm can be represented in the following way:

```
Every processor v tests that its color is unique among its neighbors
        No: select a tentative color uniformly at random from the entire
        set of colors that are not used by its neighbors

Every processor v tests that its tentative color is unique among its neighbors
        No: set color to "no color"
        Yes: set color to the tentative color
```

Figure 5.3: *Randomized Distributed Algorithm*

Distributed algorithms suit a sensor networks' application better than the sequential algorithms. There are two steps in the distributed algorithm presented above. In the first step, every node chooses a tentative colour that is unique among its neighbours. We contend that a node cannot confirm the uniqueness of its colours until after a successful broadcast. So when a node chooses a colour for the first time, this colour is only a tentative colour for that node and not a permanent one. If choice of this colour leads to a successful broadcast, then the node will make this colour its permanent colour otherwise it will again choose a new tentative colour uniformly at random. Choice of colours uniformly at random leads to the probability of two or more nodes selecting the same colour. Nodes learn that the colour was not unique through a collision that occurs due to concurrent broadcasts in the same time slot.

We propose a self-stabilizing broadcast scheduler. The algorithm is presented in Figure 5.4. Every node p_i uses a time slot $s_i \in [0, D\text{-}1]$, in a round of D time slots. Node p_i counts the number of pulses module D (the variable c_i). Node broadcasts when $c_i = s_i$. In case of collision, p_i uses a technique for uniformly selecting an empty time slot. We say that a time slot is empty from perspective, if no neighbor to pi successfully broadcasts in that time slot. In other words, a time slot is empty if no neighboring nodes broadcasts or a neighbor broadcasts a message that collides. Nodes use a single index; l_i to uniformly sample an item from an unbounded sequence. Nodes count the number t_i, of empty time slots in every round. Starting from the beginning of round, the node assigns the value of s_i to l_i and 1 to t_i. Whenever node notices that time slot $x = c_i$ is empty, it increments t_i by one and p_i assigns c_i to l_i with probability $1/t_i$.

Constants:
D = upper bound on neighboring nodes' umber
TIME_SLOT_SIZE = slot size in time units

Types:
rnd : [0,D-1] = round size
param_t = protocol parameter structure

Variables:
c: rnd = current slot in use
round:rnd = current round of communication
s:rnd = broadcast time slot
t:rnd = empty time slot counter
e:Boolean = indicates that the previous time slot is empty
b:Boolean = indicates to skip the first round when s is fresh

Macros and inlines:
`slot(t)` : *(((t) / TIME_SLOT_SIZE) mod D)*
`round(t)` : *((t) / (TIME_SLOT_SIZE * D))*

External functions:
`initParam()` : *initialization of the protocol parameters*
`Select()` : *uniform selection of an empty time slot*
`fetch()/deliver(m)`:*gets/delivers a message from/to the upper layer*
`send(m)` **:** *broadcast a message to media*
`receive(m)` : *receive a message from media*
`collHappened()` : *indicates a collision*

Upon pulse
  if e = true **then** **`MonitorEmpty()`**
  else if c = s **then** (l,t) ← (c,1)
  c ← c + 1 mod D
  if c = s **then**
    if b = **false then** **`send(fetch())`**
    (b,e) ← (**false**,**false**)
  else
    e ← **true**

Upon **`receive(m)`**
  `deliver`(m)
  e ← **false**

Upon **`collHappened()`**
  e ← **true**

Function **`MonitorEmpty()`**
  if (c = s **and** b = **false**) **then** (s,b) ← (l,**true**)
  else
    t ← t + 1
    if **`Select(`**[1,t]**`)`** = 1**then** l ← c

Figure 5.4: *The initial Self-stabilizing medium access algorithm*

6 Implementation Issues

In this section, we present the implementation details and also describe the challenges one faces when implementing a distributed sensor network implementation.

6.1 Why can't we just implement the algorithm as it is

There are many niceties that one needs to consider before implementing a protocol for sensor networks.

We see that [6] assume pair wise independent choices. By pair wise independent choice we mean that two or more nodes are likely to broadcast with equal probability and will always have the same transmit signal strength. But as we know, this assumption is hard to verify with the underlying environment. Sensor networks are characterized by unattended operation where there are harsh environmental conditions and node failures. Suppose we have two nodes A and B which are at geographically different location from a destination node C. If A and B broadcast concurrently, then the level of attenuation in their transmit signals would be different depending on their distance form C. In such scenario, the closer node would always take the bandwidth.

The existing solutions [6] and [9] use TDMA. The strict time synchronization requirement in TDMA does not sit well with the distributed nature of the sensor networks where there is a high probability of topology change as well. In distributed systems, there is no global clock or common memory. Communicating synchronized time slots across all the nodes is not straightforward.

Another issue comes when the nodes have to choose a colour. The nodes first choose a tentative colour. We call this colour tentative because a node can not confirm this to be its permanent colour until after it has been able to make a successful broadcast using this colour. One must note that colour is actually the time slot assignment for the node. So how does a node compare its tentative colour with those of its neighbours? We say that the node broadcasts using the tentative colour and then if there was a collision then it would know that this colour was also chosen by another node. Then it would choose a different colour uniformly at random. So collision detection plays the key to choosing unique colours.

As we saw in chapter 3, CSMA/CD is not useful for collision detection in senor networks. CSMA/CD is used on wired network like Ethernet because on a wire, the transceiver has the ability to listen while transmitting and so to detect collisions (with

a wire all transmissions have approximately the same strength). But, even if a radio node could listen on the channel while transmitting, the strength of its own transmissions would mask all other signals on the air [2]. So, the protocol can't directly detect collisions like with Ethernet.
Sensor networks have a bandwidth limitation. CSMA/CA techniques are hence slow and inefficient for sensor networks. CSMA/CA makes random choices before transmitting second time leaving a probability of collision even at later tries. Thus it is a probabilistic strategy that might prove too slow or inefficient for many applications.

6.2 Local testing of message arrival is hard

Local testing of message arrival can play an important role to decide if the broadcast message was successful or not. A sensor node can not test if its broadcast indeed arrive at all the other nodes because transmitting node cannot sense the channel whereas in Ethernet, when a station is ready to send, it senses the channel. In case of a free channel, it transmits data and keeps on sensing. If it does not detect any collisions then it completes its transmission successfully and concludes that the message indeed arrive without collisions. Even if it detects collision midway its transmission, it would back off.
Below we present a block diagram of how our algorithm works. There are three interesting primitives to consider in the implementation.

1. Nodes' broadcast scheduling on clock pulse: broadcast scenario
2. Nodes' action(s) on collision: collision scenario
3. Nodes' action(s) on successful receipt: reception scenario

6.2.1 Broadcast Scenario

Figure 6.1 shows the broadcast scenario. Upon every clock pulse, a node first of all checks to see if there was a collision in the previous time slot. If there was a collision and this node was involved in that collision then this node has to choose a different colour or time slot. So it chooses a new time slot uniformly at random. Nodes detect the collision by checking a local variable that is set every time there is a collision. Then the node increments the local time slots counter and checks if this value is the same as my chosen time slot number. If the local time slot counter equals the chosen slot value then the node should broadcast. Following this the node would check if it has to skip this round. If it does not need to skip the current round then it would broadcast. Why skipping the round can be helpful for fast stabilization is given in the discussion section of the report.

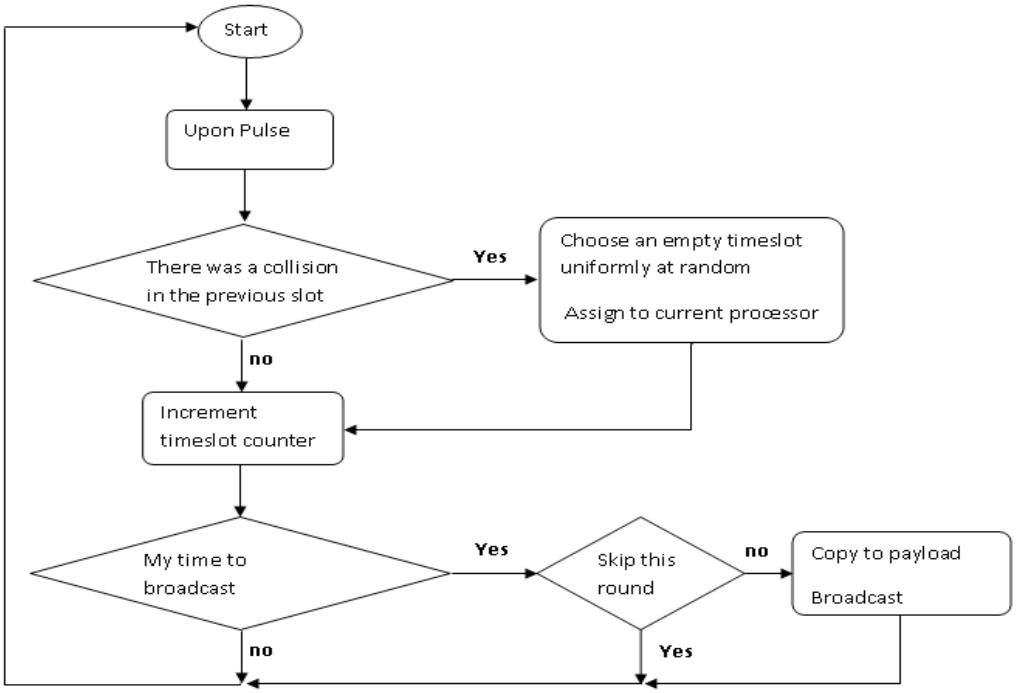

Figure 6.1: *Self-stabilizing medium access algorithm: the broadcast scenario*

6.2.2 Collision and Reception Scenarios

Figure 6.2 shows the collision scenario and the reception scenario. Our implementation contains a collision model. Before describing how we use this collision model, it is important to describe that we use TOSSIM for simulating our application. There is an event queue present in TinyOS. TOSSIM and TinyOS will be discussed in more detail in chapter. All the events are inserted in that queue indexed by time. These events are pulled from this queue by TOSSIM based on the time. Whenever there is a collision we insert the relevant collision information in the event queue. When this collision event occurs, the node marks the relevant entry in its broadcast array with collision indication. The collision model is given in Appendix A at the end of the report. Likewise in the case of a successful receipt, the node updates the relevant entry in its broadcast array with successful receipt indication. Since every node also sends its broadcast array as part of the payload. Therefore, the node also updates its broadcast array with that of the received array. Another important decision regarding slot change is also made here. If this node chose a tentative slot to broadcast but has not done so yet and receives a packet in this slot then this means that the sending node chose this slot before me. In this case, this node changes its slot and instead chooses a new slot uniformly at random. The notion of broadcast and received array are part of the revised algorithm that is discussed in detail in the following section.

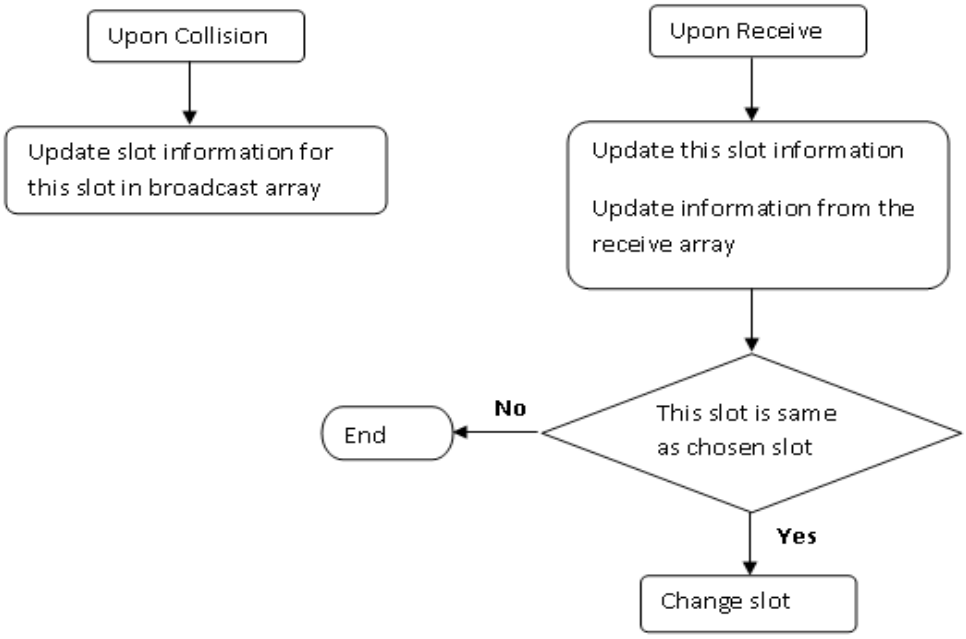

Figure 6.2: *Self-stabilizing medium access algorithm: collision and reception scenarios*

6.3 Revised Algorithm

We propose a revised self-stabilizing broadcast scheduler. We say that the stabilization becomes faster when every node has a local knowledge about the time slots of other nodes in the communication graph. For this purpose, every node maintains two arrays: a broadcast array and a received array both of size D where D is the extended degree of the communication graph. Each slot in this array contains one of the values: 0x00, 0x01, and 0x02 to indicate an empty time slot, a successful broadcast or a collision indication respectively.

The algorithm is presented in Figure 6.3. Every node p_i uses a time slot $s_i \in [0, D\text{-}1]$, in a round of D time slots. Node p_i counts the number of pulses module D (the variable $curSlot_i$). Node p_i broadcasts when $curSlot_i = s_i$. Upon successful receipt, node p_i updates information in two ways: update of this slot due to a successful receipt in that slot, and update by the received array from the other node.

It must be noted that broadcast array is part of the packet payload and p_i copies it to the payload before it broadcasts. So the received array is actually the broadcast array. This presents a solution to the hidden node problem. Since, the array p_i receives from its distance-1 neighbours contains the slot information maintained by its distance-1 neighbours about their distance-1 neighbours. This enables p_i to determine the slots chosen by its distance-2 neighbours and thus the information at p_i becomes globally more accurate.

In case of collision, p_i uses a technique for uniformly selecting an empty time slot. We say that a time slot is empty from p_i's perspective, if no neighbour to p_i successfully broadcasts in that time slot. In other words, a time slot is empty if no neighbouring nodes broadcasts or a neighbour broadcasts a message that collides. In the revised algorithm, p_i updates the broadcast array when there is collision. So nodes count the number of empty time slots in their broadcast arrays and assign to l_i a slot chosen uniformly at random.

There is another interesting case: nodes choose tentative slots; if a node chooses a time slot and then receives a broadcast in that slot before it could broadcast in that slot then it changes it slot by uniformly selecting an empty time slot. Following are two possibilities in which a node may choose a colour already used/chosen by some other node:

Case 1: It is a distance-2 neighbour, and my distance-1 neighbour has not yet reported its slot information.

Case 2: It is a distance-1 neighbour but this is its tentative colour and I have not yet received in that slot.

We see that concurrent broadcasts lead to collision and hence choice of new tentative colours. This covers the first case. The second case has two variants:

1. Both nodes chose the same tentative colour in the same round. This would result in concurrent broadcast and hence choice of new tentative colours.
2. One node chooses a tentative colour. Some other nodes choose that colour in the subsequent round since so far there has been no broadcast in that slot and that slot is considered as an empty slot. Since, the node that chose that colour first is bound to broadcast first, the receiving nodes that chose this as their tentative colour needs to change slot thereby avoiding the possibility of collisions in subsequent round and leading to faster stabilization.

The pseudo code of the revised algorithm is presented in Figure 6.3.

Constants:
N = total number of nodes in the communication graph
D = extended degree of the communication graph
TIME_SLOT_SIZE = slot size in time units
b_array[D] = broadcast array to keep track of the status of each slot
r_array[D] = received array to keep track of what is the status of slots at neighboring nodes

Types:
rnd : [0,D-1] = round size
param_t = protocol parameter structure

Variables:
curSlot: rnd = current slot in use
myRound:rnd = current round of communication
s:rnd = broadcast time slot
e:Boolean = indicates that the previous time slot is empty
b:Boolean = indicates to skip the first round when s is fresh

Macros and inlines:
`slot(t)` : *(((t) / TIME_SLOT_SIZE) mod D)*
`round(t)` : *((t) / (TIME_SLOT_SIZE * D))*

External functions:
`initParam()` : *protocol parameters initialization*
`VSelect()` : *uniform selection of an empty time slot*
`changeSlot()` *: changing slot upon successful receipt in chosen slot*
`emptySlots()` *: total available empty slots*
`assignedEmptySlot()` *: assigns an empty slot chosen uniformly at random*
`updateArray ()` : *Updates current slot information in broadcast array*
`updateArrayCmp()` : *Updates broadcast array after it receives an array from neighboring nodes*
`fetch()/deliver(m)` *:gets/delivers a message from/to the upper layer*
`send(m)` : *broadcast a message to media*
`receive(m)` : *receive a message from media*
`collHappened()` : *indicates a collision*

```
Upon pulse
  let cT = read(native_clock)
  if e = true then MonitorEmpty()
  else if curSlot = s then (l, t) ← (curSlot,1)
  curSlot ← slot(cT)
  myRound ← round(cT)
  if c = s then
     if b = false then send(fetch())
     (b,e) ← (false,false)
  else
     e ← true

Upon receive(m)
  let cT = read(native_clock)
  deliver(m)
  updateArrayCmp()
  updateArray()
  if  s  = slot(cT)  then
     changeSlot()
   e ← false

Upon collHappened()
  updateArray()
  e ← true

Function MonitorEmpty()
   if (curSlot = s and b = false ) then
      l = assignedEmptySlot(
           Vselect( emptySlots() )
           )
      (s,b) ← (l,true)

Function changeSlot()
   s = assignedEmptySlot(
         Vselect( emptySlots() )
         )
```

Figure 6.3: *The revised Self-stabilizing medium access algorithm*

7 Tools and Modifications

There are different language options available for programming for wireless sensor networks like C, DCL (Distributed Compositional Language), galsC, SNACK, SQTL, nesC. Most of these languages are extensions to C programming language. We use nesC to develop our application. We know that operating systems for sensor networks are less complex than general purpose operating systems both because of the special requirements of the sensor network applications and because of the resource constrains in sensor network hardware platform. TinyOS is the first operating system designed specifically for sensor networks. TinyOS is based on even-driven programming model. Hence, the programs are composed into event handlers and tasks. Thus, both TinyOS and the program written for TinyOS are written in nesC which is a special programming language. We use nesC because of the fact that the real motes on which the applications are to be tested come with TinyOS operating system. Following sections describe in detail the tools that we have used particularly the simulation tool TOSSIM and also present the modifications we had to make for our application.

7.1 Tools

The application is developed in nesC, a language intended for embedded systems. The empirical evaluation of the algorithm is run on the TOSSIM simulator. TOSSIM (TinyOS simulator) is a discrete event simulator for TinyOS sensor networks. Instead of compiling a TinyOS application for a mote, one can compile it into the TOSSIM framework, which runs on a PC. This allows debugging, testing, and analyzing algorithms in a controlled and repeatable environment. As TOSSIM runs on a PC, one can examine the TinyOS code using debuggers and other development tools. TinyOS's event-driven execution maps well into a discrete event simulator. The entire application is event driven where hardware interrupts are modeled as simulator events. TOSSIM complies directly from TinyOS source and replaces hardware with software components.
There is an event queue in TOSSIM, where events are inserted prioritized by time. Time is maintained as simulation ticks per second. There are 4 x 106 simulation ticks per second. Network in TOSSIM is modelled in the form of a communication graph. When TOSSIM runs, it pulls events of the event queue (sorted by time) and executes them. There are two programming interfaces to TOSSIM; python and C++. In our implementation we have used the python interface.

By default, no nodes can communicate with other nodes in TOSSIM. In order to simulate the network behaviour we have to specify a network topology. We can specify how we want the underlying network to be laid. We can specify the entire connectivity of the communication graph; fully connected or partially connected. Moreover we can specify the gain values on each link. Gain is the magnitude of attenuation in the transmit signal strength when it reaches the receiver. The default values for TOSSIM's radio model are based on the CC2420 radio, used in the micaZ, telos family. It uses an SNR curve derived from experimental data collected using two micaZ nodes, RF shielding, and a variable attenuator.
TOSSIM also simulates the RF noise and interference a node hears, both from other nodes as well as outside sources. It uses the Closest Pattern Matching (CPM) algorithm. CPM takes a noise trace as input and generates a statistical model from it. This model can capture bursts of interference and other correlated phenomena, such that it greatly improves the quality of the RF simulation.
The default MAC object has a large number of functions, for controlling the back off behaviour, packet preamble length, radio bandwidth, etc. All time values are specified in terms of radio symbols, and one can configure the number of symbols per second and bits per symbol. By default, the MAC object is configured to act like the standard TinyOS 2.0 CC2420 stack: it has 4 bits per symbol and 64k symbols per second, for 256kbps. One can change the back off behaviour using a subset of the MAC functions.
It is possible to inject packets in the running network with TOSSIM. This is useful when one wants to communicate the degree of the communication graph to the entire set of nodes. TOSSIM also allows inspecting variables which is useful to control the simulation. For example, we want to stop the simulation when all nodes start to broadcast successfully in each round. Each node has only a local knowledge which it can convey by setting a local variable. We can stop the simulation when all nodes in the communication graph have set their variables.

7.2 Modifications

We make the following modifications to TOSSIM.

1. We modify the default packet-level radio component in TOSSIM. This component by default implements a basic CSMA algorithm. In this algorithm a node transmits if and only if it measures a clear channel min_free_samples () in a row. One can set the value of min_free_samples which currently is 2. The node sample up to max_iterations () times. If it does not detect a free channel in this time, it reports channel busy and the send is not successful. Otherwise it transmits followed by a back off period. We completely remove the back off behaviour. In our implementation, a mote sends information on

the data link layer regardless of the fact that the channel might be busy (i.e. when it has something to send, and is able to send from the transport layer). Then, we use the receiver side collision detection to report collisions. And thus nodes choose their slots accordingly (uniformly at random) on next pulse.

2. We incorporate a collision model in the interference model of TOSSIM. We insert a collision indication event in the event queue under the following scenarios.
    a. A mote loses a packet from another mote due to being in the midst of reception.
    b. A mote loses a packet when it receives a concurrent stronger packet. In our implementation, we have that a node starts successfully hearing packet A and during packet A it starts to hear a much stronger packet B. While B's preamble could be detected, the radio stack just sees it as a corruption of packet A. Therefore A is lost and B is not received.

It should however be noted that this indication is done on the receiver side and it is the receiver who reports this collision to the other nodes in the communication graph. Collision information has the following structure:

```
struct collision_info {
      int sender;
      int receiver;
      int slot;
      bool collision_flag;
      sim_time_t coll_time;
      struct collision_info* next;
};
```

3. By default, TinyOS limits the frame payload to 28 bytes. For our revised algorithm we needed a bigger packet size since we pass the broadcast array in the payload. We do so by adding the following in our Makefile

    CFLAGS += -DTOSH_DATA_LENGTH=xx
    where xx is the desired payload size.

8 Preliminary Numerical Evaluation

We model the system as one that consists of a set of communicating entities, which we call processors (or motes/nodes). We assume that our clocks are synchronized. In this regard, all motes use a common source of global pulses available in TOSSIM. We also assume that our collision model is able to detect all collisions on media access layer.

8.1 The Communication and Collision Graphs

The real sensor network applications comprise a large number of sensor nodes where each mote can communicate with a limited number of other nodes due to their geographical distribution. This motivates to use those communication graphs in our simulation which are only partially connected. We employ the concept of random geometric graphs. A random geometric graph is a random undirected graph generated by placing vertices at random uniformly and independently on the region. The two vertices are connected if and only if the distance between them is at most a threshold r. Though the graphs are partially connected but each node is linked with the other nodes through a bounded number of hops.
Collision graph on the other hand means a set of those nodes in the communication graph whose concurrent broadcasts would lead to collision due to their being in the immediate neighbourhood of each other. Our graphs are symmetric meaning that if there is a communication link between nodes A and B then A communicates with B and B can communicate with A. Moreover we specify uniform gain values on all the links. Figure 8.1 represents an example communication graph used in our simulations.

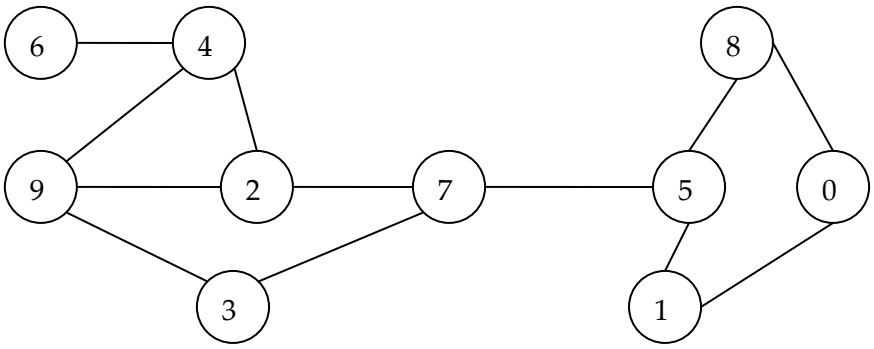

Figure 8.1: *An example random geometric communication graph of ten nodes*

8.2 Different Sets of Graphs and Number of Repetitions

We conduct experiments with four set of nodes comprising five, ten fifteen and twenty nodes. We use the following methodology to calculate results.
For each set of nodes, we generate four random graphs. For every graph, we carry out five simulations each of two hundred communication rounds. This makes a total of twenty experiments against each set of nodes. Our result is averaged first for the five simulations of one graph. Since we have four different graphs, the result is further averaged for the four graphs.
For comparison purposes we also run the experiments with the existing basic back off algorithm in TOSSIM. We use two terms "imposed congestion" and "diffused congestion". Imposed congestion means that when the algorithm starts, all motes broadcast in the same time slot. This starting configuration leads to all broadcast colliding. Diffused congestion on the other hand means that nodes make a random choice of slots when the algorithm starts. This leads to nodes broadcasts that are distributed over the whole communication round. We start our algorithm with an imposed congestion and the basic back off algorithm with a diffused congestion. Clearly, this configuration gives the basic back off algorithm an advantage over our algorithm with regards to number of collisions. Since, our algorithm is self-stabilizing we see that despite starting in an arbitrary configuration, eventually all nodes broadcast successfully in all communication rounds.
The following section details what data we calculate followed by a discussion of how our algorithm performs in comparison to the basic back off algorithm.

8.2.1 Messages and Data grams

We calculate the number of transport layers messages delivered by each node referred to as messages and physical datagram messages received from each node referred to as data grams. Transport layer message is the message broadcast by a

mote. This message is sent from the transport layer to the media access layer. Datagram message is the physical datagram that each mote who is in the communication range receives on its media access layer. For each message, a datagram is sent on all the links of the mote.

8.2.2 Transport Layer Messages

Figure 8.2 shows the percentage of transport layer messages delivered by each set of motes in two hundred communication rounds. We see that slotted algorithm shows nearly hundred percent throughputs. The fact that throughput is not hundred percent is due to the message losses by collisions before stabilization. On the other hand, back off algorithm could only deliver sixty percent messages. It is interesting to note that the algorithm perform consistently good with increasing size of the network.

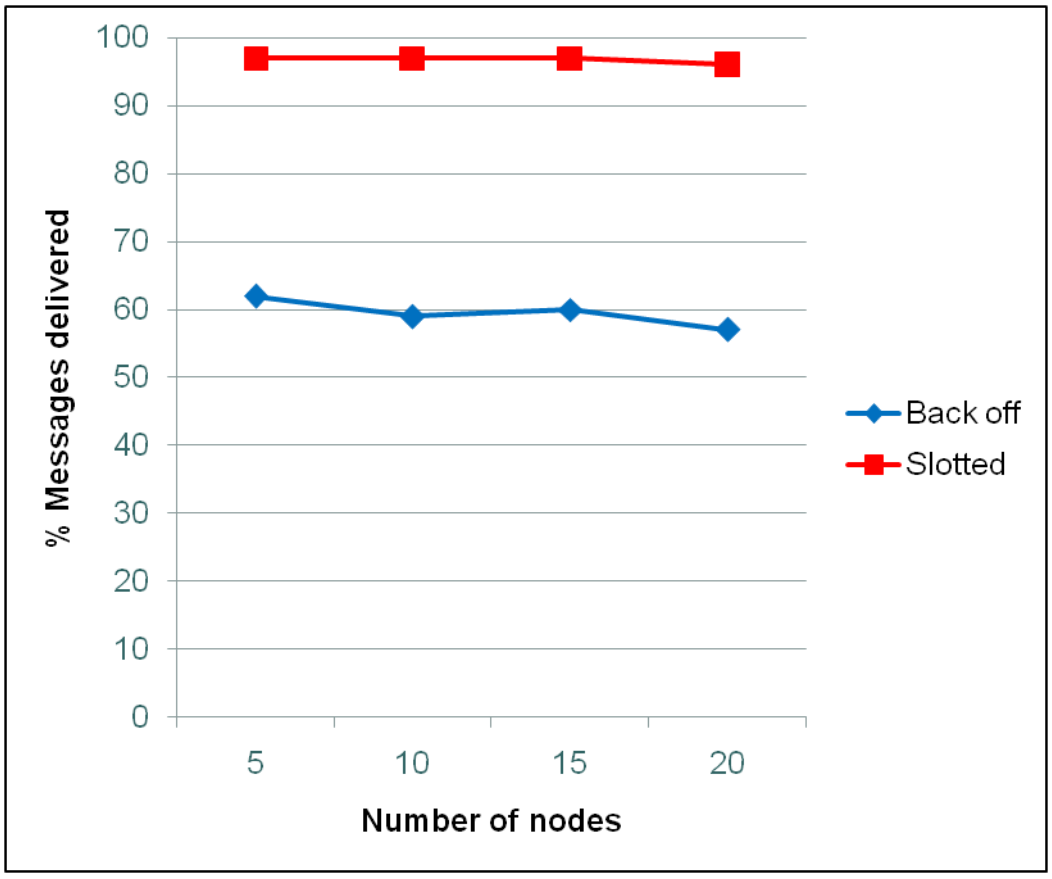

Figure 8.2: *The percentage of transport layer messages delivered with slotted algorithm and back off algorithm*

8.2.3 Datagram Messages

Figure 8.3 shows percentage of message received on the media access layer with each set of motes. It must be noted that these are the messages received on the media access layer and not the messages delivered. Slotted algorithm would pass all messages from the transport layer over to media access layer only after stabilization. Since increasing number of nodes increases the possibility of collisions, there is a small decline in the throughput. With back off algorithm, no decision regarding passing a message to media access layers is made on the transport layer. Hence, back off algorithm indicates almost hundred percent received data grams.

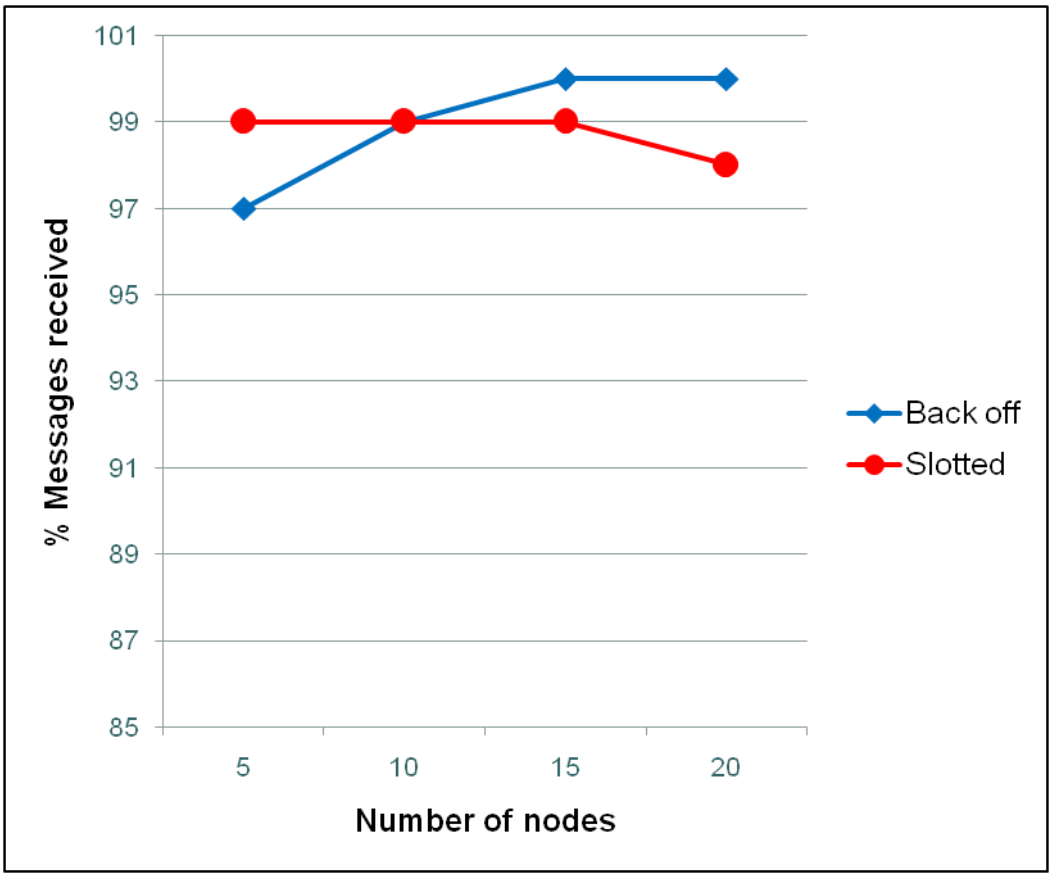

Figure 8.3: *The percentage of data gram messages received with slotted algorithm and back off algorithm*

From Figure 8.2 and Figure 8.3, one can see that with back off algorithm nearly hundred percent data grams are received but only about sixty percent messages are delivered. This means that the basic back off algorithm results in about forty percent of messages being lost in collisions.

8.2.4 Lost Messages

Figure 8.4 shows the number of lost messages in both the algorithms. The messages are lost primarily due to collisions on the media access layer. One can see that as the size of the network increases, there is an increase in the number of lost messages as well. The slotted algorithm shows a consistent pattern in which there is only a small increase in the number of lost messages with increased size of the network, whereas in the back off algorithm the number of lost messages is significantly large. It should be noted that this computation was carried out in a simulation of two hundred communication rounds.

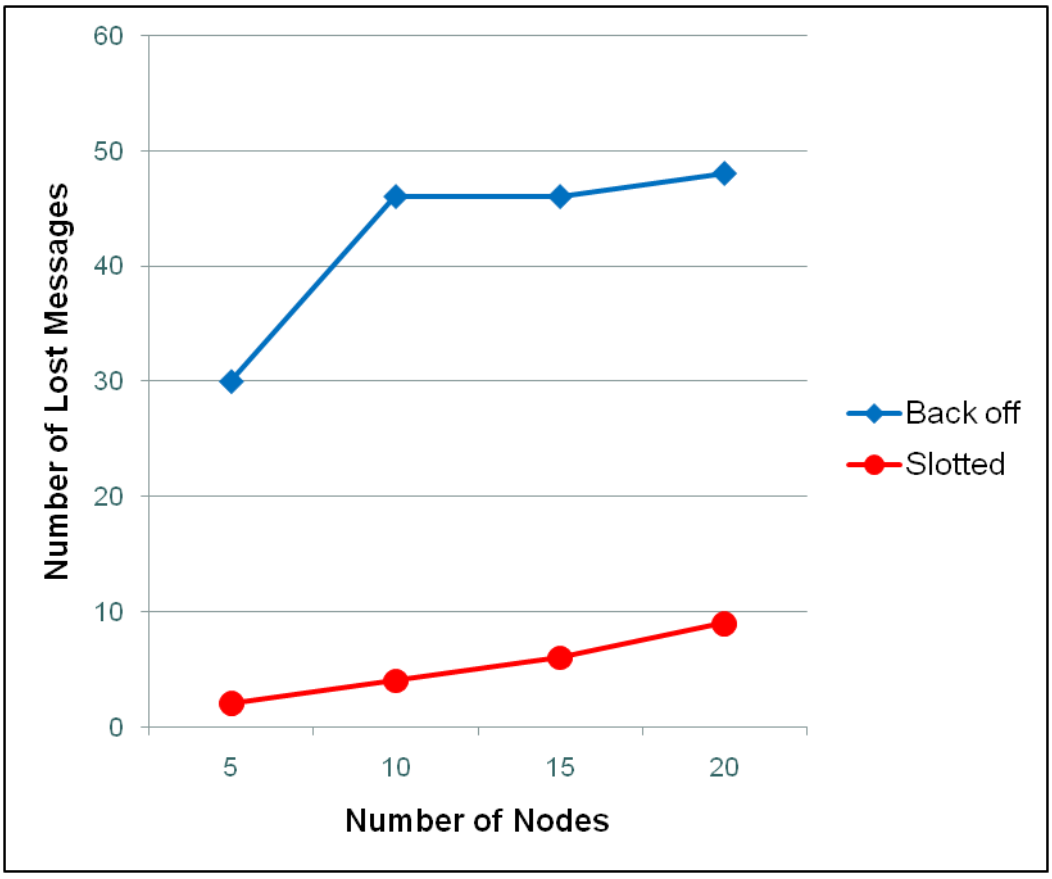

Figure 8.4: *The number of lost messages with slotted algorithm and back off algorithm*

8.2.5 Stabilization Time

Stabilization time is the number of communication rounds after which all nodes in the sensor network are guaranteed to broadcast successfully in each subsequent round of communication. Figure 8.5 shows the stabilization time curve. We can see that with an increase in the number of nodes, the curve shows only a small increase. For our experiments, this has been a very interesting result as for a network of as large as thirty five nodes, the average stabilization time was never more than fourteen communication rounds. One should note that our algorithm allows a large number of nodes to broadcast successfully even before stabilization.

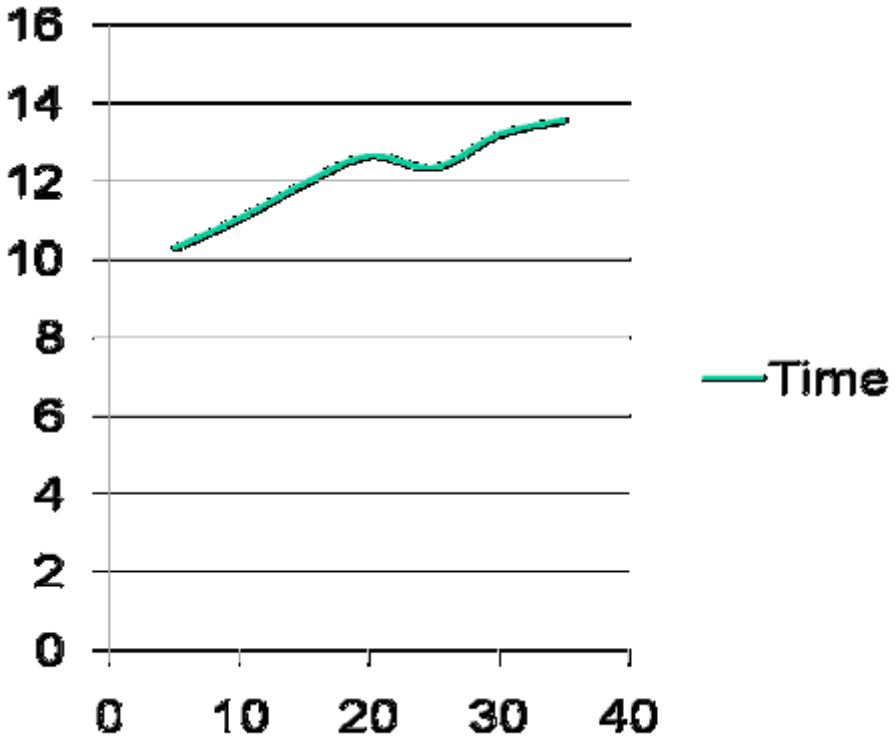

Figure 8.5: *The stabilization time for networks of different sizes in a simulation of two hundred communication rounds*

8.2.6 Stabilizing Behaviour

The charts presented in the following figures are for the developed media access algorithm only since there is no notion of stabilization in the back off algorithm. Figure shows that the slotted algorithm quickly stabilizes and a large portion of the messages is delivered after the stabilization.

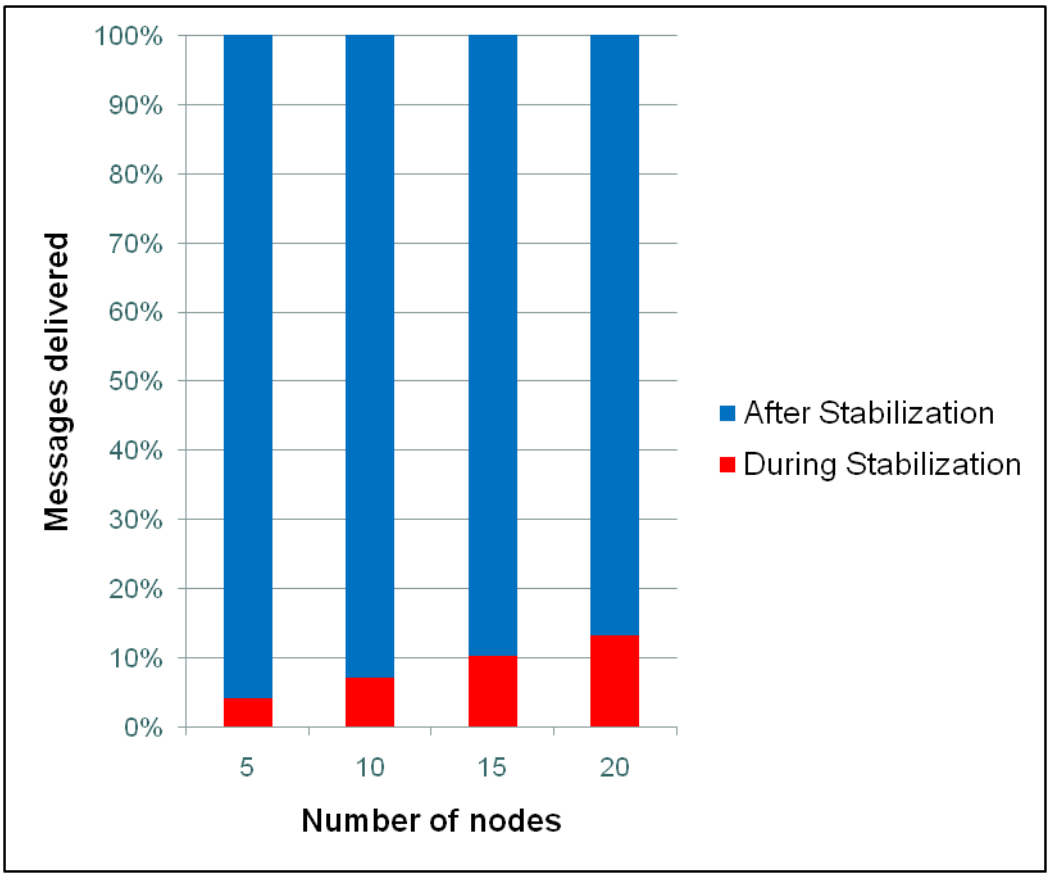

Figure 8.6: *The percentage of transport layer messages delivered with slotted algorithm during and after stabilization*

Figure 8.7 shows the data grams received during and after the stabilization with slotted algorithm. Again we can see the quick stabilization. An important thing to note is that unlike the back off algorithm there is no random choice of broadcast time in our algorithm. Hence, there is always a chosen slot; fixed or tentative in which a node would broadcast and eventually all the nodes are allocated individual slots to broadcast. Therefore, during stabilization, the data grams are received for tentative or fixed slot broadcasts and not all the time. This contributes to showing a small decrease in the number of data grams received in slotted algorithm as compared to the back off algorithm as shown in Figure 8.3.

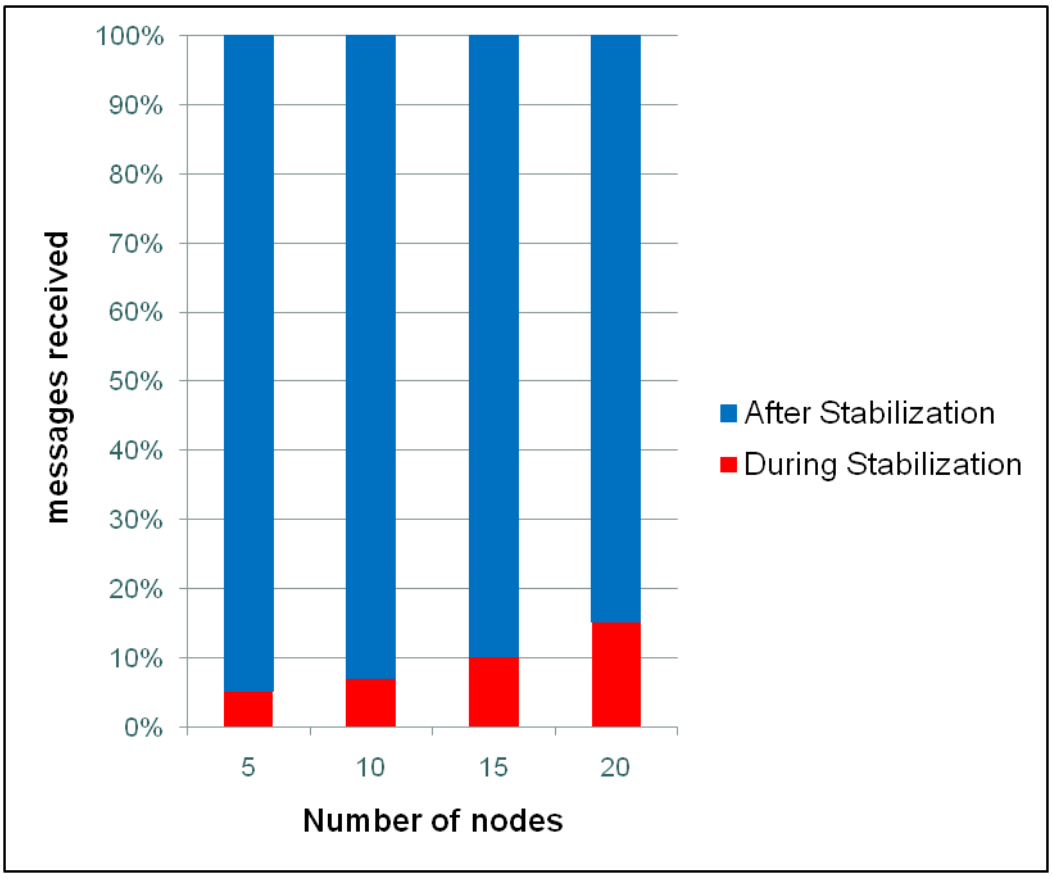

Figure 8.7: *The percentage of datagram messages received with slotted algorithm during and after stabilization*

Figure 8.8 shows that there are no message losses after the stabilization. It is interesting to see that during stabilization too, a large number of nodes are able to broadcast with success.

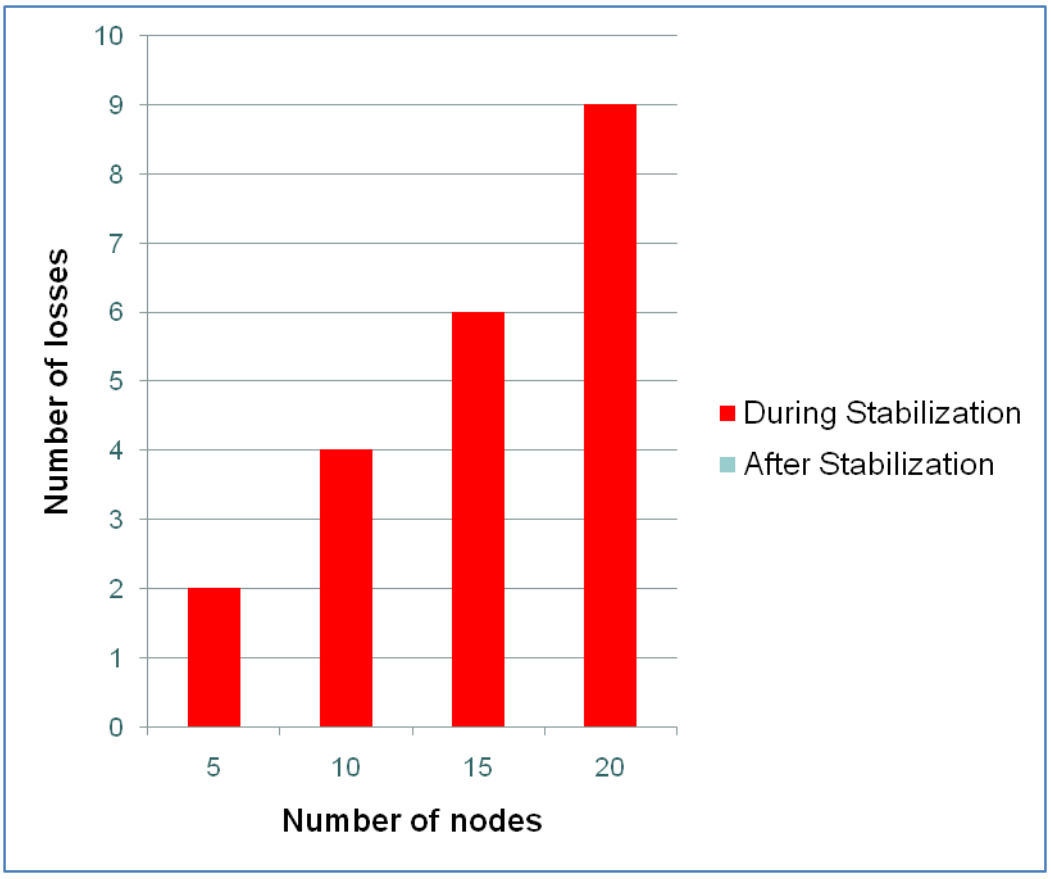

Figure 8.8: *The number of lost messages received with slotted algorithm during and after stabilization*

9 Discussion

Media access algorithms in sensor networks have the direct influence on the eventual energy conservation. Collision detection techniques play a vital role in media access algorithm design. Traditional carrier sense techniques for collision avoidance such as CSMA/CA suffer from message delays and long stabilization times. Such techniques cannot perform well under saturated situations. There are other techniques such as CSMA/CD which are suitable only for wired networks such as Ethernet. [9] Use TDMA for media access. The strict time synchronization requirement in TDMA does not sit well with the fact that topology of the sensor network might change. Under such situations, we see that receiver side collision detection seems a useful option to report collision in sensor networks. Moreover, it is the receiver which realizes the collision making it a natural choice for sensor networks.

We assume an ideal environment that has access to synchronized clocks. We say that the algorithm can be extended to account for the clock skews of a realistic environment.

We get a stabilization time curve that is not linear and increases very slightly with increasing network size. We say that the stabilization can be faster when the size of the broadcast array is same as the extended degree of the interference graph. Extended degree is the sum of me, my immediate neighbours and the neighbours of my immediate neighbours all taken uniquely or in other words the number of nodes in my distance two neighbourhood? The values in a node's broadcast array are its view of the network state. We know that, it is sufficient for a node to test the distance two neighbourhood in order to choose a time slot that will be safe for successful broadcast. A smaller size of the broadcast array can actually lead to faster stabilization. In small networks extended degree equals the graph diameter. But on large random geometric graphs, a node's extended degree can be much smaller than the actual diameter of the interference graph. Thus we say that our algorithm can perform consistently well for large networks and hence is scalable.

In our algorithm we skip a round when the node's chosen a colour is fresh. In a distributed setting a node can only choose a colour that is tentative i.e. it is not final until after the node has had a successful broadcast using that colour. Skipping the round after choosing the colour helps stabilize faster. If the node receives some packet successfully in its chosen slot and in the round that it is not broadcasting (skipped round), then this implies that another node already chose this colour and was able to broadcast with success. In order to avoid collisions in subsequent rounds, this node then changes it colour by choosing a new colour uniformly at random.

The following two sections describe the extensions that might be added to the developed algorithm and our conclusions from doing this project.

9.1 Future Work

To make stabilization faster, the idea of local stabilization can be employed. Since we use the randomized geometric graphs, the geographical distribution of the nodes reduces interdependence among different network portions. Hence certain portions of the networks can get stabilized before other network portions. In the current setting, stabilization time is the number of rounds after all the nodes in the network has a stabilized view of the network. Since certain applications might require the network to be stabilized for them to work properly. In such instance, nodes would be able to declare local stabilization and continue to start such applications.
While currently there is a uniform bandwidth allocation to all the nodes, the algorithm can be made communication adaptive. Thus, a larger bandwidth may be allocated to the nodes which are required to participate in the communication more often depending on the needs of the underlying application.
The communication adaptation can be taken on the same lines as in [6]. Nodes can elect a local leader. This leader would then allocate bandwidth depending upon the communication requirements of the nodes.

9.2 Conclusions

We carry out the empirical evaluation of the algorithm using the TOSSIM simulator. We develop a media access algorithm that does not use the carrier sensing and back off behaviour on the media access layer and instead uses the receiver side collision detection. We observe that; there is a rapid stabilization of the network; the networks can be of varying sizes and random geometries, a large number of nodes are able to broadcast even during the stabilization, The throughput of the network is nearly hundred percent and there is only a small overhead after stabilization. Hence, after the empirical evaluation carried out using a large number of simulations as described in chapter 8, we contend that the distributed algorithm works using receiver-side collision detection.
Stabilization time is $O((n + log\ n))$ rounds. This stabilization time is calculated by running simulation for a network of as large as thirty five nodes. $log\ n$ is the dominating factor in experiments; this means that the algorithm is scalable to larger network sizes.
The algorithm is tolerant towards overheads. The algorithm performs well even under saturated situations where datagram messages are always available. Given such starting configuration, the nodes undergo quick stabilization. Then after the

stabilization, the overhead is smaller under such situations as compared to similar algorithms such as [6] and [8].

Appendix A

Collision Model

```
// CollisionModel.nc


#include ''coll.h''


interface CollisionModel{
	command void set_collision(collision_info_t** headRef, int se, int r,
int sl, bool v, sim_time_t t);
	event void collHappened(collision_info_t* c);
}


//*************************************************************************
//*************************************************************************


// coll.h


#ifndef COLL_H_INCLUDED
#define COLL_H_INCLUDED


#include <sim_gain.h>


struct collision_info {
	int sender;
	int receiver;
	int slot;
	bool collision_flag;
	sim_time_t coll_time;
	struct collision_info* next;
};


typedef struct collision_info collision_info_t;
sim_event_t* allocate_collision_event(sim_time_t t, collision_info_t* c);
collision_info_t* allocate_coll_info();
void sim_gain_collision_handle(sim_event_t* evt);









/*************************************************************************
Parts of UscGainIntereferenceModelC.nc which implements CollisionModel.nc
*************************************************************************/
```

```
// Setting the collision parameters
command void CModel.set_collision(collision_info_t** headRef, int sender, int
receiver, int slot, bool val, sim_time_t tm)
{
	collision_info_t* newCollision =
	collision_info_t*)malloc(sizeof(collision_info_t));
	newCollision->sender = sender;
	newCollision->receiver = receiver;
	newCollision->slot = slot;
	newCollision->collision_flag = val;
	newCollision->coll_time = tm;
	newCollision->next = *headRef;
	*headRef = newCollision;
}


// the collision handler
void sim_gain_collision_handle(sim_event_t* evt){
	int id;
	collision_info_t* temp = (collision_info_t*)evt->data;
	id = evt->mote;
	signal CModel.collHappened(temp);
}


// allocating the collision event.
sim_event_t* allocate_collision_event(int node_id, sim_time_t t){
	sim_event_t* evt = (sim_event_t*)malloc(sizeof(sim_event_t));
	evt->mote = node_id;
	evt->time = t;
	evt->handle = sim_gain_collision_handle;
	evt->cleanup = sim_queue_cleanup_event;
	evt->cancelled = 0;
	evt->force = 1;
	evt->data = head;

	return evt;
}


/*************************************************************************
Part of TestC.nc which uses CollisionModel.nc
*************************************************************************/


event void CModel.collHappened(collision_info_t* temp){
	uint16_t slot;
	uint32_t cT;
	if(temp==NULL)
		printf("\nEND OF LIST\n");
	else
		slot = temp->slot;
		cT = call Timer0.getNow();
		collVar++;
		updateArray(0, slot, 2);
		prm.e = TRUE;
}
```

Bibliography

[1]Bhaskar Krishnamachari. *Networking Wireless Sensors*

[2] *Linux Wireless LAN Howto.* Available: http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Linux.Wireless.mac.html

[3] HyungJune Lee, Alberto Cerpa, Philip Levis. *Improving Wireless Simulation through Noise Modeling*

[4] Kurtis Kredo, Prasant Mohapatra. *Medium Access Control in Wireless Sensor Networks*

[5] Dongjin Son, Bhaskar Krishnamachari, John Heidemann. *Experimental Study of Concurrent Transmission in Wireless Sensor Networks*

[6] Ted Herman, Sebastien Tixeuil. *A Distributed TDMA Slot Assignment Algorithm for Wireless Sensor Networks*

[7] Murat Demirbas, Srivats Balachandran. *RobCast: A Singlehop Reliable Broadcast Protocol for Wireless Sensor Networks.*

[8] Gregory Chockler, Murat Demirbas and Seth Gilbert. *Consensus and Collision Detectors in Wireless Ad Hoc Networks*

[9] Khaled A. Arisha, Moustafa A. Youssef* and Mohamed F. Younis*. *Energy-Aware TDMA-Based MAC for Sensor Networks*